Generating Monotone Trees : Extended Abstract (Theoretical Computer Science and its Applications)

AUTHOR(S):

Kusakari, Yoshiyuki; Niisato, Yoshimi; Notoya, Jyunichi; Kasai, Masao

数理解析研究所講究録 1426 巻 2005 年 172-177

Generating Monotone Trees (Extended Abstract)

草苅良至†, 新里善美‡, 能登谷淳一† 笠井雅夫†
Yoshiyuki Kusakari†, Yoshimi Niisato‡, Junichi Notoya†, and Masao Kasai†
†秋田県立大学システム科学技術学部電子情報システム学科
Department of Electroncs and Information Systems,
Faculty of Systems Science and Thechnology, Akita Prefectural University
‡秋田県立大学大学院システム科学技術研究科電子情報システム学専攻
Department of Electroncs and Information Systems,
Graduate School of Systems Science and Thechnology, Akita Prefectural University

1 Introduction

A *linkage* is a collection of line segments, called *bars*, possibly joined at their ends, called *joints*. A *reconfiguration* of a linkage is a continuous motion of their bars, preserving the length of each bar and disallowing bars to cross. A reconfiguration of a linkage is called *planar* if all bars are in $\mathbb{R}^2$ during the motion. In this paper, we consider only a planar reconfiguration of a planar linkage, and we may omit the word "planar." Recently, such reconfiguration problems have been attracted special attention[3, 6, 8].

For such planar reconfiguration, there is a fundamental question; whether any tree linkage can be *flattened*. The answer of this question is "no" since there exist several "locked trees" which can not be flattened in $\mathbb{R}^2$ [1, 2, 4, 7, 8]. For instance, Figure 1 illustrates a locked tree[2]. It is desired to characterize the class of trees which can be flattened. Recently, Kusakari *et. al.* found a class of trees, called *monotone trees*, in which any tree can be flattened [6]. Figure 2 illustrates a monotone tree[6]. Kusakari *et. al.* show that any monotone tree can be flattened, and give a method for flattening monotone trees[6]. Thus, the monotonicity is the sufficient condition for the class of trees which can be flattened.

In this paper, we give a randomized algorithm for generating various kinds of monotone trees. For characterizing monotone tree T, there are may parameters, such as the number $|J|$ of joints, the distribution of degrees, and the distribution of bar lengths, $\cdots$. In this paper, we focus on the number $|J|$ of joints. Thus, the input of our algorithm is the natural number $n \in \mathbb{N}$, and the output is a monotone tree $T = (J, B)$ having n joints and $n-1$ bars. Our algorithm generate a monotone tree of n joints in time $O(n \log n)$, using space $O(n)$.

2 Preliminaries

Let $L = (J, B)$ denote a linkage consisting of a joint set J and a bar set B. A structural graph of a linkage L is denoted by $G(L)$. An embedding of a structural graph $G(L)$ is called a *configuration* of linkage L.

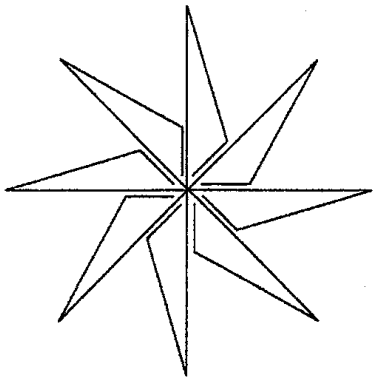

Figure 1: A locked tree[2].

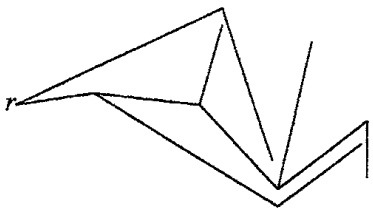

Figure 2: A monotone tree [6].

A linkage $L = (J, B)$ is simple if nonadjacent bars $b, b' \in B$ do not cross each other. In this paper, we consider only a simple linkage, and we may omit the word "simple."

A linkage L is called a *(rooted) tree linkage* or a *(rooted) tree* if the structural graph $G(L)$ is a (rooted) tree. Let $T = (J, B)$ be such a rooted tree linkage, and $r \in J$ be the root of T. For convenience, with out loss of generality, we may assume that the root r has the minimum x-coordinate. The set of children of joint $j \in J$ is denoted by $N(j) \subset J$, and the set of descendant of joint $j \in J$ is denoted by $D(j) \subset J$. The *degree* $d(j)$ of a joint j is the number $|N(j)|$ of neighbors of j. A *leaf* is a joint having no children. A join j is *inner* if j is neither the root r nor a leaf. A linkage L is called a *chain* if the structural graph $G(L)$ is a path. A chain C is *monotone (in x direction)* if the intersection of C and any vertical line is either a single point or empty. A tree T is *monotone(in x direction)* if every root-leaf chain in T is monotone in x direction.

For any point $p \in \mathbb{R}^2$, the x-,y- coordinate of p is denoted by $x(p)$,$y(p)$, respectively. For any vertical line l or any vertical line segment s, the x- coordinate is denoted by $x(l)$,$x(s)$, respectively. The crossing point of two segments s, s' is denoted by $c(s, s')$. For a point $p \in \mathbb{R}^2$ and a direction $d \in (-\pi, \pi]$, the *ray* starting from p and going in the direction d is denoted by $r_d(p)$. In this paper, we may denote by $d = -y, x, y, -x$ as the direction $d = -\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi$, respectively.

3 An algorithm for Generating Monotone Tree

In this section, we give an algorithm for generating a monotone tree.

3.1 Properties of Monotone Tree

For a monotone tree, the following lemma holds:

Lemma 1 *A tree $T = (J, B)$ is a monotone tree if and only if the following two condition holds:*

(i)(monotonicity) *for any joint $j \in J$ and any descendant $j' \in D(j)$,*

$$x(j) \leq x(j');$$

(ii)(simplicity) *any two bar $b, b' \in B$ do not cross each other .*

(Proof) Obvious. □

We will design a randomized algorithm for generating monotone trees, whose correctness is based on this Lemma 1. Moreover, our algorithm technically based on so called the plane sweep methods [9]. Thus, our algorithm generates a monotone tree constructing joints from the root to leaf by sweeping the plane $\mathbb{R}^2$ from left to right. Note that, on the plane sweep method, it does explicitly not need that the geometrical data in ahead (right) side of the sweep line S. We apply this property of the plane sweep method to design our algorithm. Thus, our algorithm adequately keeps the bars which intersect the sweep line S,and are called *sweeping bars.*

3.2 Algorithm

In this subsection, we design an generating algorithm.

We use two data structures:

(1) the *sweeping bars* SB, in which elements $b_i \in SB$ are ordered by the y-coordinate of crossing points $c(b_i, S)$; and

(2) the *event schedule* ES, which keeps two kinds of *event*, called *joint event* and *crossing point event.*

Joint Event $JE(j)$: A joint event $JE(j)$ corresponds to a joint j and a incident bar $b = (j', j)$, where $x(j') < x(S) < x(j)$. Any joint event is executed when the sweep line S reached $x(j)$.

Crossing Point Event $CE(b,b')$: A crossing point event $CE(b,b')$ corresponds to a crossing point $c(b,b')$ of successive bars $b,b' \in SB$, where $x(S) < x(c(b,b'))$. Any crossing point event $CE(b,b')$ is executed when the sweep line S reached $x(c(b,b'))$.

We denote by $x(E)$ the x-coordinate of event E, which is either a joint event or a crossing point event. We can adequate update the event schedule ES so that events E in ES are ordered by the x-coordinate. Note that the order of events by execution may be different from the order of events by generation.

Our algorithm is a randomized algorithm with arguments the number $n \in \mathbb{N}$ of joints and the maximum degree $\Delta \in \mathbb{N}$, and generates a monotone tree $T = (J,B)$ with $|J| = n$ joints in which the degree $d(j)$ of any joint $j \in J$ is at most Δ.

Algorithm Generate Monotone Tree: GMT(n,Δ)
Let $B := \phi, J := \{r\}, SB := \phi, ES := \{JE(r)\}, x(S) := -\infty$.

Step1: Get the event E with the minimum x-coordinate from ES,and remove E from ES;

Step2: Move the sweep line S to $x(E)$, thus let $x(S) := x(E)$;

Step3: Execute adequate event depends on the kind of E, thus execute either a joint event JE or a crossing point event CE;

Step4: Go back to Step 1 if $|J| < n$.

(Joint Event $JE(j)$)
The incident bar $b = (j',j)$ is in SB (if $j \neq r$). Let b_a be the bar successive above b in SB, and let b_b be the bar successive bellow b in SB. Remove b from SB, insert $b = (j',j)$ into B, and insert j into J. Next, we choose degree $d(j)$ at random so that $1 \leq d(j) \leq \Delta$ if $ES = \phi$, or $0 \leq d(j) \leq \Delta$ if $ES \neq \phi$. Then, the following two cases occur:
{Case1:$(d(j) = 0)$} Check whether b_a and b_b cross each other, and then insert $CE(b_a,b_b)$ to ES if they cross.
{Case2:$(d(j) \geq 1)$} Generate new $d(j)$ bars $b_1 = (j,j_1), b_2 = (j,j_2), \cdots, b_{d(j)} = (j,j_{d(j)})$, whose slopes and lengths are selected at random, so that $x(j) \leq x(j_i)$. (See Lemma 1(i).)

Note that the bar set B is not updated at this time. With out loss of generality, we may assume that the indices are sticked on bars by the order of bars lighted by the ray $r_d(j)$ from $d = -\frac{\pi}{2}$ to $d = \frac{\pi}{2}$. Then, insert sweeping bars $b_1, b_2, \cdots, b_{d(j)}$ into SB so that the order is kept, and insert joint events $JE(j_1), JE(j_2), \cdots, JE(j_{d(j)})$ into ES. Furthermore, insert $CE(b_a,b_1)$ into ES if b_a and b_1 cross each other,and insert $CE(b_{d(j)},b_b)$ into ES if $b_{d(j)}$ and b_b cross each other. Figure 3 illustrates a joint event.
(Crossing point event $CE(b,b')$)
For two bars $b = (j_l,j_r)$ and $b' = (j'_l,j'_r)$, without loss of generality, we assume that $y(j_l) > y(j'_l)$ and $y(j_r) < y(j')$. Let b_a be the bar successively above b in SB, and b_b be the bar successively below b' in SB. Then, either b or b' is chosen at random, and the chosen bar generate no descendant. We call such chosen bar a *cut bar*. We assume that b' is such a cut bar. Then, remove b' from SB,and $JE(j'_r)$ form ES. Furthermore, remove all crossing point event $CE(b',b'')$ for any pair of $b',b'' \in SB$ if they remain ES. Next, we choose a scale $s \in (0,1)$ at random. Then, generate a new bar $b^* = (j'_l,j^*)$ so that $|b^*| = s \times |j'_l c(b,b')|$, and insert b^* into B and insert j^* into J.

We may call the process above *cut*. One can easily observe that the cut operation ensures simplicity of the tree. (See Lemma 1(ii).) Figure 4 illustrates a crossing point event.

3.3 Analysis

In this subsection, we analyze the algorithm $GMT(n,\Delta)$.

Let $T = (J,B)$ be a tree generated by $GMT(n,\Delta)$. Then,$|J| = n$ and $|B| = n-1$. Moreover, let $J_i \subset J$ be the set of inner joints, and $J_l \subset J$ be the set of leaves. Any inner joint $j_i \in J_i$ is generated by a joint event. On the other hand, a leaf is generated by the following (a),(b),or(c):

(a) Zero is chosen for the degree $d(j)$ at joint event $JE(j)$;

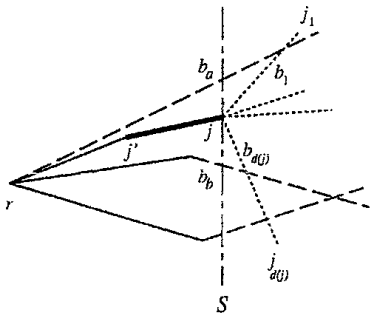

Figure 3: Joint event $JE(j)$.

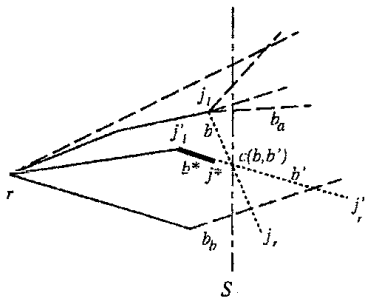

Figure 4: Crossing point event $CE(b, b')$.

(b) A bar is cut at a crossing point event $CE(b, b')$;

(c) When n joints are generated, then the algorithm $GMT(n, \Delta)$ is terminated,and all joint events remained in ES are removed. The parent joints of these removed joint events make leaves of T.

For $p \in \{a, b, c\}$, let $J_{l_p} \subset J$ be the set of joints generated by above (p). Then, the following equation holds.

$$\begin{aligned} |J| &= |J_i| + |J_l| \\ &= |J_i| + |J_{l_a}| + |J_{l_b}| + |J_{l_c}| \\ &= n. \end{aligned} \tag{1}$$

Let $\mathcal{E}$, $\mathcal{JE}$, and $\mathcal{CE}$ be the set of events E, joint events JE, and crossing point events CE, respectively, which are generated during the execution of $GMT(n, \Delta)$. On the other hand, there are three kinds of event. Let $\mathcal{E}^e$, $\mathcal{E}^c$, and $\mathcal{E}^r$ be the set of events executed during $GMT(n, \Delta)$, removed by cuts, and remained ES at the end of the algorithm, respectively. Then, the following equation holds.

$$\begin{aligned} |\mathcal{E}| &= |\mathcal{JE}| + |\mathcal{CE}| \\ &= |\mathcal{E}^e| + |\mathcal{E}^c| + |\mathcal{E}^r|. \end{aligned} \tag{2}$$

Then,the following lemmas holds for the number of events, the proofs of which are omitted in this extended abstract.

Lemma 2 *The number $|\mathcal{JE}|$ of joint events is $O(n)$.*

Lemma 3 *The number $|\mathcal{CE}|$ of crossing point events is $O(n)$.*

Lemma 4 *The number $|\mathcal{E}|$ of events is $O(n)$.*

Thus, the following theorem holds, the proofs of which are omitted in this extended abstract.

Theorem 1 *$GMT(n, \Delta)$ generate a monotone tree $T = (J, B)$ with n joints in time $O(n \log n)$, using space $O(n)$.*

4 Experimental Results

We implemented our algorithm $GMT(n, \Delta)$ in previous section by java language. For the implementation, we add parameters other than the parameters n and Δ of the algorithm, such as the maximum length m of bars, and the range $[s_{min}, s_{max}]$ of slopes of bars. When we execute the joint event $JE(j)$, we generate the new bars $b_i = (j, j_i)$ so that the bar length $|b_i|$ is uniformly chosen from $(0, m]$ at random,

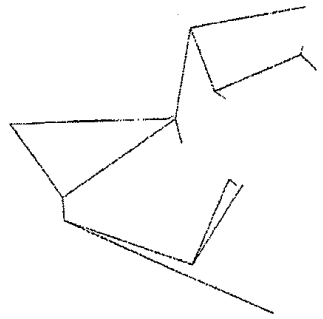

Figure 5: An example of the output obtained by $GMT(20, 2)$.

and the slope $s(b_i)$ of bar b_i is uniformly chosen from $[s_{min}, s_{max}]$ at random. Figure 5 illustrates an example of the output obtained by $GMT(n, \Delta)$, each parameters of which is assigned as follows:

$$\begin{aligned} n = 20, \Delta = 2, m = 10, \\ s_{min} = -0.8 \times \frac{\pi}{2}, s_{max} = 0.8 \times \frac{\pi}{2}. \end{aligned} \tag{3}$$

We give additional definition for evaluating monotone trees.

The *(properly) inner part* of line segment s is the open line segment obtained by removing both ends from s. The line segment s is *visible in d direction from a point $p \in \mathbb{R}^2$ (for a linkage $L = (J, B)$)* if the ray $r_d(p)$ pass through the inner part of s and does not intersect any bars in B. For two line segments s, s', s is *visible in d direction from s'* if the inner part of s' has a point p from which s is visible in d-direction. Obviously, s is visible in d direction from s', if and only if s' is visible in $(d + \pi)$ direction from s. The *visible pair* $vp(b, b')$ is the pair of bars $b, b' \in B$ such that $b \in B$ is visible in x direction from $b' \in B - \{b\}$. The number of the visible pairs of bars $b, b' \in B$ mainly depends on the time complexity of the algorithm for flattening monotone trees in [6]. Thus, one may observe that the number of visible pairs is significant feature of monotone tree. Therefore, we evaluate the number of visible pairs for the monotone trees generated by our algorithm. For a tree $T = (J, B)$, we denote by $VP(T)$ the set of visible pairs. We denote by $\widetilde{VP(n)}$ the maximum $|VP(T)|$ for the any trees $T = (J, B)$ of n joints. For the upper limit of the number of visible pair, the following lemmas holds, the proofs of which are omitted in this extended abstract.

Lemma 5 *Let K be the number of joints which is visible in x or $-x$ direction from the point at infinity. Then, the following equation holds:*

$$\widetilde{VP(n)} \leq \frac{10n - 4K - 13}{3}.$$

On the experiment, the parameters other than n are fixed the values as follows:

$$\begin{aligned} \Delta = 2 \text{ or } 3, m = 10, \\ s_{min} = -0.8 \times \frac{\pi}{2}, s_{max} = 0.8 \times \frac{\pi}{2}. \end{aligned} \tag{4}$$

Then, we generate 10 monotone trees of n joints from $n = 10$ to 100 by 10, and evaluate the average number of visible pairs $\overline{VP(n)}$ of n joint linkage. Figure 6 shows the the average number of visible pairs for the case $\Delta = 2$, and Figure 7 shows for the case $\Delta = 3$. From these figures, one can observe that the number of visible pairs linearly increase for the number of joints, and does not mainly depends on the maximum degree.

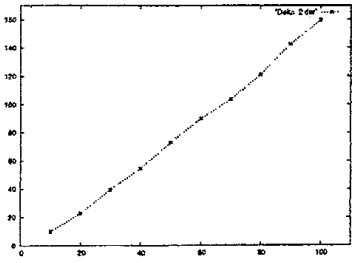

Figure 6: The average number of visible pairs($\Delta = 2$).

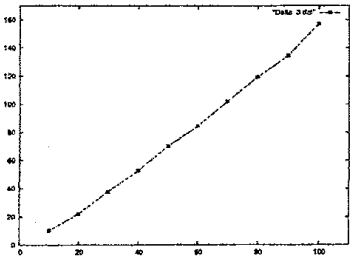

Figure 7: The average number of visible pairs($\Delta = 3$).

5 Conclusion

In this paper, we design an algorithm $GMT(n, \Delta)$ for generating monotone tree $T = (J, B)$ with $n = |J|$ joints, and the degree of each joint $j \in J$ is at most Δ. Our algorithm uses so called plane sweep technique, and runs in time $O(n \log n)$ if the generate tree has n joints. Furthermore, We implemented our algorithm by java language.

The following future works remain.

(i) From the Figures 6 and 7, one can observe that the coefficient of linear term is between $1.4 \sim 1.6$. However, the theoretical upper bound of the coefficient is $\frac{10}{3}$. Thus, there is a large difference between them. Find the reason of this difference.

(ii) Find another class of trees in which tree can be flattened other than monotone tree.

References

[1] H. Alt,C. Knauer,G. Rote,and S. Whitesides, "The Complexity of (Un)folding," Proc. 19th ACM Symp. Computational Geometry, pp.164–170,2003.

[2] T. Biedl,E. Demaine,M. Demaine,S. Lazard, A. Lubiw,J. O'Rourke, S. Robbins,I. Streinu, G. Toussaint,and S. Whitesides, "A note on reconfiguring tree linkages: Trees can lock," Discrete Applied Mathematics, vol.254,pp. 19–32,2002.

[3] R. Connelly,E. Demaine, and G. Rote, "Straightening polygonal arcs and convexifying polygonal cycles," Proc. IEEE Symp. Foundations of Computer Science, pp. 432–442,2000.

[4] R. Connelly, E. Demaine, and G. Rote, "Infinitesimally Locked Self-Touching Linkages with Applications to Locked Trees," Physical Knots: Knotting, Linking, and Folding Geometric Objects in R^3, American Mathematical Society, pp. 287–311,2002.

[5] R. Diestel,Graph Theory,Springer-Verlag,1997.

[6] Y. Kusakari, M. Sato, and T. Nishizeki, "Planar Reconfiguration of Monotone Trees," in IEICE Trans. Fund., Vol.E85-A, No.5, pp.938–943,2002.

[7] Y. Kusakari, "On Reconfiguring Radial Trees," Proc. JCDCG2002, Lecture Notes in Computer Science, vol.2866, Springer-Verlag, pp.182-191,2003.

[8] Y. Kusakari, Y. Niizato, J. Notoya, and M. Kasai, "An Algorithm for Generating Monotone Trees(in Japanese)," Manuscript.

[9] F.P. Preparata, M.I. Shamos, Computational Geometry:An Introduction, Springer-Verlag, 1985.